A Tsume-Shogi Processor Based on Reconfigurable Hardware

Yohei Hori[†], Tsutomu Maruyama[‡] and Kenji Toda[†]

[†] *Information Technology Research Institute,*
National Institute of Advanced Industrial Science and Technology
1-1-1, Umezono, Tsukuba, Ibaraki, 305-8568 Japan
E-mail: {hori.y, k-toda}@aist.go.jp

[‡] *Institute of Engineering Mechanics and Systems, University of Tsukuba*
1-1-1, Ten-nou-dai, Tsukuba, Ibaraki, 305-8573 Japan
E-mail: maruyama@darwin.esys.tsukuba.ac.jp

Abstract

High performance, low cost and compact specialized hardware for tsume-shogi *(shogi problems) has been developed with a Field-Programmable Gate Array (FPGA). Developing dedicated hardware systems is an essential approach to improve the playing strength of shogi programs. However, inflexibility and high cost of hardware have been significant problems in development of the systems. An FPGA gives solutions to the problems of hardware implementation. To devise parallel and pipeline architecture of shogi hardware and test the feasibility of an FPGA for shogi, we first implemented a tsume-shogi solver. With the latest FPGA, we successfully implemented all of the highly parallelized modules on a single chip. The hardware tsume-shogi solver achieved about 11 times higher performance than software on Pentium4-2.53 GHz. The devised architecture can be also applied for normal shogi.*

1 Introduction

Shogi [1], or Japanese chess, is a challenging target in artificial intelligence for game playing. The complicated rules and large search space have been motivating researchers to study shogi programs. The complexity of shogi is mainly attributed to the *reuse* of captured pieces. In shogi, the pieces captured from the opponent can be put back onto the board (this type of move is called a *drop*). The reuse of pieces causes a large number of legal moves, and consequently the game tree of shogi becomes enormously large. The number of nodes in the game tree of chess and shogi are reportedly 10^{123} and 10^{226} respectively [3].

To date, lots of algorithms to deal with the complicated rules and large search space have been developed, nevertheless shogi programs are not competent enough to beat human professional players. To improve the playing strength of shogi programs, developing specialized hardware systems is an essential approach. However, inflexibility and high cost of hardware are the serious problems in developing shogi hardware.

Using a Field-Programmable Gate Array (FPGA) helps in resolving these problems in shogi hardware development. With an FPGA, favorite logic circuits are realized by downloading configuration data from a host computer, and the implemented circuits can be easily changed by downloading data again. So new algorithms can be implemented and tested as soon as they are devised. Additionally, the price of an FPGA is much lower than that of typical hardware devices when ordering a few test chips.

Tsume-shogi is a mating problem in shogi. Developing a tsume-shogi solver is effective to devise parallel and pipeline architecture of shogi hardware and test the feasibility of an FPGA for shogi. As already reported by the authors, FPGAs are quite effective for move generation and elimination in tsume-shogi [1, 2]. This time we implemented a tree search algorithm and realized a complete tsume-shogi processor with a single FPGA. We solved actual tsume-shogi problems and compared the performance of hardware with software. In this paper, we show the architecture of the modules and discuss the performance of the tsume-shogi processor.

[1] For brief explanation of the rules of shogi and difference between shogi and chess, see [4].

2 Hardware Algorithms

The main ideas for fast move generation and size reduction of hardware are parallel computation of different types of moves and parallel/pipeline processing with the 9-square architecture. This section briefly explains the hardware algorithms to generate check/defense moves and to search the game tree.

2.1 Check/Defense Move Generation

Ideally, computation with hardware should be performed to all 81 (9x9) squares simultaneously in shogi. In that case, however, complicated wiring causes a decline of hardware performance. So our modules deal with 9 squares (1 rank) at a time to generate check/defense moves. Although it takes 9 clocks to scan the whole board (9 ranks) with this method, hardware works at high frequency because of simple wiring. As data of the whole board are processed in pipeline, the computation of shogi with 9-square architecture finishes in quite short time.

In our method, check and defense moves are categorized into 3 groups respectively, therefore totally 6 move generators are in the hardware. Check categories are *Direct*, *X-ray* and *Drop*, and defense categories are *King-Evasion*, *Defensive-Capture* and *Blockade*. Each move generator has the 9-square architecture, and all generators work in parallel. The detail procedure for move generation, architecture of the move generators and data structures are described in [1, 2].

2.2 Tree Search

The implemented tree search algorithm is PN* introduced by Seo [8]. Seo's tsume-shogi solver is the first program to solve the longest problem with the answer of 1525 steps. In hardware, tree search is controlled with a complicated state machine. Figure 1 shows the state transition diagram of the tree search circuit. The details of the state transition are out of the scope in this paper. The detail structures of the tree search circuit are described in Section 3.

2.3 The Block Diagram

Figure 2 shows the block diagram of the move generator. The number described on each module corresponds to the pipeline stage of computation of move generation. All modules described in Figure 2 work in parallel, and additionally, parallel data processing of 9 squares and fine-grained pipeline computation are performed in all modules. Therefore all groups of moves are generated in quite short time.

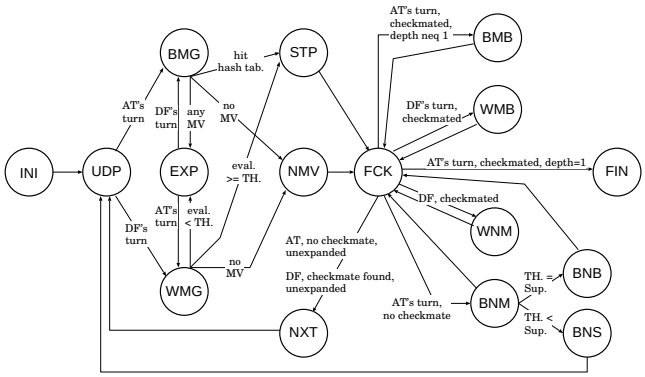

Figure 1. The state transition diagram of Tree Search Circuit.

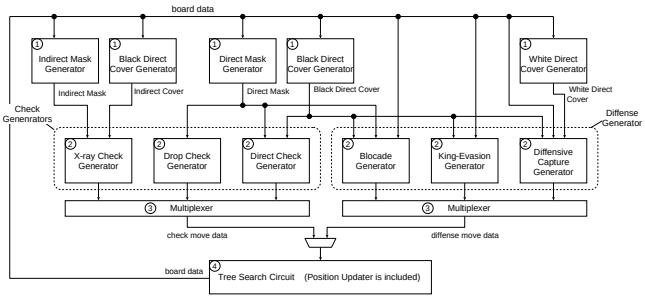

Figure 2. The block diagram of the tsume-shogi hardware

3 Tree Search Circuit

3.1 Data Structure

The structure of data used in the circuit depends on the implemented tree search algorithm, but it is independent of the structure of check move, defense move or other data generators. Therefore various kinds of tree search algorithms can be implemented without changing the structure of move generators. This feature is useful for testing the feasibility of different tree search algorithms.

Moves generated in the move generators are all sent to Tree Search Circuit and stored to a stack. Deepening and backtracking of the game tree is realized by controlling the stack. The implemented algorithm uses iterative deepening, so the hash table is used to avoid expanding the same node again. Here we explain the structure of the move stack and the hash table.

3.1.1 Move Stack *Move Stack* is a 16 k x 38 bit memory and is constructed with internal memory of an FPGA. Two

Table 1. The data structure of Hash Table.

name	width (bit)	explanation
Data Flag	1	Occupancy (stored/empty).
Evaluation	8	Evaluation of the position.
Depth	11	Depth of the mate-found node.
Compressed Pos.	88	Prevents malfunction caused by hash collision.

kinds of data are stored to *Move Stack*. One is *Move Data* sent from *Check/Defense Move Generator*. Move data are stored to Move Stack with information about depth of the node and mate flag. The other is information to control tree search. After the move generation in some position finished, information about the number of generated moves, the upper limit of the threshold in the iterative deepening and address of previous move is stored to the Move Stack.

3.1.2 Hash Table In iterative deepening, the same node may be evaluated and expanded many times. To avoid the idle evaluation of the node, hash table is used in our circuit. Data stored to *Hash Table* are evaluation of the node, the depth of mate-found node and compressed board data. Compressed board data are referred when the hash collision occurred. When the compressed code of the current position is different from the stored code and the evaluation of the current position is smaller than stored value, stored hash code is overwritten with the current one. In addition, exceptional operation is done when the stored position is a mate-found node. A mate-found node is not overwritten to speed up tree search.

Hash Table is a 1 M x 108 bit of external memory. Table 1 shows the structure of *Hash Table*. If more memory resource is available, larger hash table is preferable to avoid hash collision.

3.2 The Structure of the Circuit

Figure 3 shows the structure of *Tree Search Circuit*. Board Stack is a memory where board data are stored. The stored data are used to avoid re-calculating board data when backtracking the tree. Hash Code Stack is a memory where hash code data are stored. The stored data are used to avoid re-calculating hash code when backtracking the tree.

4 Implementation and Performance

4.1 Implementation Results

We are currently porting the tsume-shogi hardware to a new hardware platform *REX2* [9] vended by REXEON Technology, Inc [7]. REX2 has an XC2VP70-6 [10]

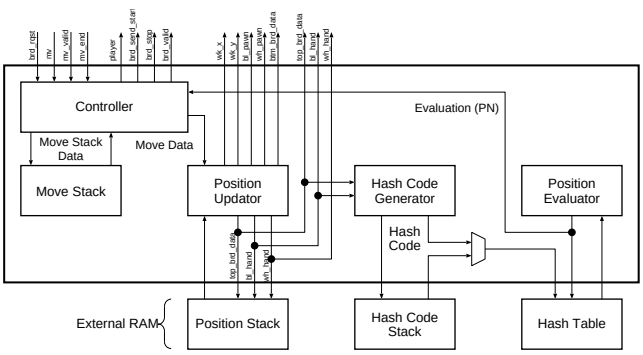

Figure 3. The structure of Tree Search Circuit.

Table 2. The implementation result of each module.

Module	Slice	Block RAM	Freq. [MHz]
Check Move	11,977 (36%)	66 (20%)	112.108
Defense Move	7,147 (21%)	25 (8%)	93.879
Tree Search	1,080 (3%)	43 (13%)	140.262
TOTAL	20,204 (61%)	134 (41%)	93.879

(33,088 slices, 328 Block RAMs), 18MB DDR2-SSRAM and up to 2GB DDR-SDRAM.

Below we show the result of the implementation of modules in Table 2. Table 2 shows that the implemented modules totally use 41% of slices and 61% of Block RAMs, and the whole circuit works at 93.879 MHz. Figure 4 describes the timing chart of the tsume-shogi solver. Figure 4 shows that required clock cycles for move generation is $54 + N$. N is the number of valid moves by the attacker or the defender in a position. In all modules, required clock cycles except N is always constant and does not depend on the number of moves to be generated.

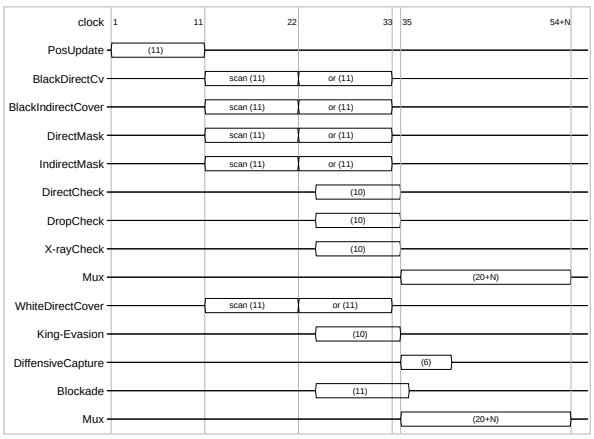

Figure 4. The timing chart of move generation in the tsume shogi hardware.

Table 3. The number of the nodes expanded in the tsume-shogi problems.

steps	# of prbs.	# of nodes	SW [ms]	HW [ms]	ratio
3	2	1940	55	2.6	20.9
5	9	5494	137	10.2	13.5
7	17	3282	80	5.1	15.8
9	23	12452	270	22.2	12.2
11	23	19901	379	33.8	11.8
13	17	28389	546	48.5	11.2
15	6	60889	1012	89.7	11.3
17	1	37801	590	56.5	10.4
overall	98	19874	330	29.4	11.2

4.2 Performance

We selected 98 tsume-shogi problems from the book "Naito's Tsume-Shogi Selection" [6] as benchmark problems. Table 3 shows the average number of expanded nodes and the computation time taken to solve tsume-shogi with software and hardware. Software was run on Pentium 4–2.53 GHz + Cygwin (Unix emulator on Windows 2000). Software is specially designed for this benchmark test, and employs the same tree search algorithm and transposition table as hardware does. Software is designed to perform completely the same move generation and ordering as hardware, though some software algorithms are different from hardware ones to gain the best computation speed. The performance of hardware is calculated on the assumption that the clock frequency is 93.879 MHz.

As shown in Table 3, our hardware works 11.2 times faster than our software and is able to generate about 676,000 positions per second on average. For comparison, the performance of one of the most powerful software (Nagai's program, run on Athlon 1.4 GHz) is 100,000–120,000 nodes per second [5].

Here we make further discussion on the performance of hardware. As the frequency of the circuit is $93.879\,\mathrm{MHz}$ and required clocks is $54 + N$, computation time for generating N moves in a position (T_N) is $(54 + N) \times (1/93.879)$ [μsec] . As found out from this expression, computation time for the move generation in hardware does not significantly depend on the number of generated moves because of highly parallelized architecture of the circuit. For example, T_5 and T_{10} are 0.628 and 0.682 μsec respectively, that is, T_{10} is only 1.08 times longer than T_5 in hardware while that in software would be about twice longer. Therefore the circuit can work efficiently for a tsume-shogi problem where lots of possible moves are to be generated. Usually in tsume-shogi puzzles, the number of moves generated in a position is limited because all pieces are seldom used and the attacker is given minimum piece-in-hand to solve the problem. Our hardware would show higher performance in practical end-game positions in normal shogi.

5 Conclusions

In this paper, we presented a high performance, low cost and compact tsume-shogi solver based on a Field-Programmable Gate Array (FPGA). We categorized check and defense moves into 6 groups (3 groups each) and generated moves of all categories in parallel to gain high performance. Modules in each move generator are implemented parallel and pipeline architecture to achieve high speed computation. With the latest FPGA, we successfully implemented all tsume-shogi modules on a single chip.

We solved 98 tsume-shogi problems with hardware and software and compared the performance. As a result, hardware turned out to work about 11 times faster than software. The performance of hardware against software depends on the game tree complexity. Hardware is expected to show better performance in more complex tsume-shogi problems and practical end-game positions in normal shogi.

6. References

[1] Y. Hori, M. Seki, R. Grimbergen, T. Maruyama, and T. Hoshino. A shogi processor with a field programmable gate array. In *Computers and Games*, pages 297–314, 2000.

[2] Y. Hori, M. Sonoyama, and T. Maruyama. An FPGA-based processor for shogi mating problems. In *IEEE International Conference on Filed-Programmable Technology*, pages 117–124, 2002.

[3] H. Matsubara. Algorithms of move generation in a shogi program. In *Game Programming Workshop*, pages 134–138, 1994.

[4] H. Matsubara and R. Grimbergen. Differences between shogi and western chess from a computational point of view. In *Board Game in Academia*, 1997.

[5] A. Nagai. *Df-pn Algorithm for Searching AND/OR Trees and Its Applications*. PhD thesis, University of Tokyo, Tokyo, Japan, 2001.

[6] K. Naito. *Naito Tsume-Shogi Selection*. Japan Shogi Association, Tokyo, Japan, 2002. in Japanese.

[7] REXEON Technology, Inc. http://www.rexeon.com/.

[8] M. Seo, H. Iida, and J. W. H. M. Uiterwijk. The PN*-search algorithm: Application to tsume-shogi. *Artificial Intelligence*, 129(1–2):253–277, 2001.

[9] K. Toda and K. Sayano. Porting of M32R, the soft-macro processor, to REX, the multiprocessor experimental platform. Technical Report CPSY2003-51, The Institute of Electronics, Information and Communication Engineering, Tokyo, March 2004. In Japanese.

[10] Xilinx, Inc., San Jose, CA. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet v4.0*, June 2004.